

lib/main/nat-less.ath

```

1 # Ordering properties of natural numbers
2
3 load "nat-times"
4
5 extend-module N {
6
7 declare <: [N N] -> Boolean [[int->nat int->nat]]
8
9 module Less {
10 open Plus
11
12 #.....
13 # Define strict inequality on type N
14
15 assert* <-def := [(zero < S n)
16                 (~ n < zero)
17                 (S n < S m <==> n < m)]
18
19 define [zero<S not-zero injective] := <-def
20
21 # assert zero<S      := (forall n . zero < S n)
22 # assert not-zero   := (forall m . ~ (m < zero))
23 # assert injective := (forall m n . S m < S n <==> m < n)
24
25 define [n' x y z] := [?n':N ?x:N ?y:N ?z:N]
26
27 #.....
28
29 define <S          := (forall n . n < S n)
30 define =zero      := (forall n . ~ zero < n ==> n = zero)
31 define zero<     := (forall n . n /= zero <==> zero < n)
32 define S1        := (forall x y . S x < y ==> x < y)
33 define S2        := (forall x y . x < y ==> x < S y)
34 define S4        := (forall m n . S m < n ==> exists n' . n = S n')
35 define S-step    := (forall x y . x < S y & x /= y ==> x < y)
36 define discrete  := (forall n . ~ exists x . n < x & x < S n)
37 define transitive := (forall x y z . x < y & y < z ==> x < z)
38 define transitive1 := (forall x y z . x < y & ~ z < y ==> x < z)
39 define transitive2 := (forall x y z . x < y & ~ x < z ==> z < y)
40 define transitive3 := (forall x y z . ~ y < x & y < z ==> x < z)
41 define irreflexive := (forall n . ~ n < n)
42 define asymmetric := (forall m n . m < n ==> ~ n < m)
43 define S-not-<    := (forall n . ~ S n < n)
44 define Reverse-S := (forall n m . ~ m < n ==> n < S m)
45 define trichotomy := (forall m n . ~ m < n & m /= n ==> n < m)
46 define trichotomy1 := (forall m n . ~ m < n & ~ n < m ==> m = n)
47 define trichotomy2 := (forall m n . m = n <==> ~ m < n & ~ n < m)
48 define Plus-cancellation := (forall k m n . m + k < n + k ==> m < n)
49 define Plus-k          := (forall k m n . m < n ==> m + k < n + k)
50 define Plus-k1         := (forall k m n . m < n ==> m < n + k)
51 define Plus-k-equiv    := (forall k m n . m < n <==> m + k < n + k)
52 define not-equal      := (forall m n . m < n ==> m /= n)
53 define not-equal1     := (forall m n . m < n ==> n /= m)
54
55 } # Less
56
57 #.....
58
59 declare <=: [N N] -> Boolean [[int->nat int->nat]]
60
61 module Less= {
62
63 define [n' x y z] := [?n':N ?x:N ?y:N ?z:N]
64
65 assert* <==def := [(x <= y <==> x < y | x = y)]
66 #assert <==def := (forall x y . x <= y <==> x < y | x = y)
67
68 define definition := <==def

```

```

69
70 define Implied-by-<      := (forall m n . m < n ==> m <= n)
71 define Implied-by-equal := (forall m n . m = n ==> m <= n)
72 define reflexive      := (forall n . n <= n)
73 define zero<=        := (forall n . zero <= n)
74 define S-zero-S-n    := (forall n . S zero <= S n)
75 define injective     := (forall n m . S n <= S m <==> n <= m)
76 define not-S         := (forall n . ~ S n <= n)
77 define S-not-equal   := (forall k n . S k <= n ==> k /= n)
78 define discrete      := (forall m n . m < n ==> S m <= n)
79 define transitive    := (forall x y z . x <= y & y <= z ==> x <= z)
80 define transitive1   := (forall x y z . x < y & y <= z ==> x < z)
81 define transitive2   := (forall x y z . x <= y & y < z ==> x < z)
82 define S1           := (forall n m . n <= m ==> n < S m)
83 define S2           := (forall n m . n <= m ==> n <= S m)
84 define S3           := (forall n . n <= S n)
85 define trichotomy1  := (forall m n . ~ n <= m ==> m < n)
86 define trichotomy2 := (forall m n . ~ n < m ==> m <= n)
87 define trichotomy3 := (forall m n . n < m ==> ~ m <= n)
88 define trichotomy4 := (forall m n . n <= m ==> ~ m < n)
89 define trichotomy5 := (forall m n . m <= n & n <= m ==> m = n)
90 define Plus-cancellation :=
91     (forall k m n . m + k <= n + k ==> m <= n)
92 define Plus-k        := (forall k m n . m <= n ==> m + k <= n + k)
93 define Plus-k1       := (forall k m n . m <= n ==> m <= n + k)
94 define k-Less=       := (forall k m n . n = m + k ==> m <= n)
95 define zero2         := (forall n . n <= zero ==> n = zero)
96 define not-S-zero    := (forall n . ~ S n <= zero)
97 define S4           := (forall m n . S m <= n ==> exists n' . n = S n')
98 define S5           := (forall n m . n <= S m & n /= S m ==> n <= m)
99 define =zero        := (forall m . m < one ==> m = zero)
100 define zero<=one    := (forall m . m = zero ==> m <= one)
101 } # Less=
102
103 #.....
104 # Proofs
105
106 by-induction Less.<S {
107   zero => (!chain-> [true ==> (zero < S zero) [Less.zero<S]])
108 | (S n) => (!chain-> [(n < S n) ==> (S n < S S n) [Less.injective]])
109 }
110
111 datatype-cases Less.=zero {
112   zero =>
113     assume (~ zero < zero)
114     (!reflex zero)
115 | (S n) =>
116     assume A := (~ zero < S n)
117     (!from-complements (S n = zero)
118       A
119     (!chain-> [true ==> (zero < S n) [Less.zero<S]]))
120 }
121
122 conclude Less.zero<
123 pick-any n
124   (!equiv
125     conclude (n /= zero ==> zero < n)
126     (!contra-pos (!instance Less.=zero [n]))
127     conclude (zero < n ==> n /= zero)
128     assume (zero < n)
129     (!by-contradiction (n /= zero)
130       assume (n = zero)
131       (!absurd
132         (!chain-> [(zero < n) ==> (zero < zero) [(n = zero)]])
133         (!chain-> [true ==> (~ zero < zero) [Less.not-zero]])))
134
135 by-induction Less.S1 {
136   zero =>
137     conclude (forall ?y . S zero < ?y ==> zero < ?y)
138   pick-any y

```

```

139   assume Less := (S zero < y)
140   (!two-cases
141     assume (y = zero)
142     (!by-contradiction (zero < y)
143       assume (~ zero < y)
144       let {not-Less :=
145         conclude (~ S zero < y)
146         (!chain->
147           [true ==> (~ S zero < zero) [Less.not-zero]
148             ==> (~ S zero < y) [(y = zero)]])
149         (!absurd Less not-Less))
150       assume nonzero := (y /= zero)
151       let {has-predecessor :=
152         (!chain-> [nonzero ==> (exists ?m . y = S ?m)
153                   [nonzero-S]])
154         pick-witness m for has-predecessor
155         (!chain->
156           [true ==> (zero < S m) [Less.zero<S]
157             ==> (zero < y) [(y = S m)]])
158 | (S n) =>
159   conclude (forall ?y . S S n < ?y ==> S n < ?y)
160   let {induction-hypothesis := (forall ?y . S n < ?y ==> n < ?y)}
161   pick-any y
162   assume Less := (S S n < y)
163   (!two-cases
164     assume (y = zero)
165     (!by-contradiction (S n < y)
166       assume (~ S n < y)
167       let {not-Less :=
168         (!chain->
169           [true
170             ==> (~ S S n < zero) [Less.not-zero]
171             ==> (~ S S n < y) [(y = zero)]])
172         (!absurd Less not-Less))
173       assume nonzero := (y /= zero)
174       let {has-predecessor :=
175         (!chain->
176           [nonzero
177             ==> (exists ?m . y = S ?m) [nonzero-S]])
178         pick-witness m for has-predecessor witnessed
179         (!chain->
180           [(S S n < y)
181             ==> (S S n < S m) [witnessed]
182             ==> (S n < m) [Less.injective]
183             ==> (n < m) [induction-hypothesis]
184             ==> (S n < S m) [Less.injective]
185             ==> (S n < y) [(y = S m)]])
186 }
187
188 # It's simpler if we use datatype-cases for the case splitting:
189
190 by-induction Less.S1 {
191   zero =>
192     datatype-cases (forall ?y . S zero < ?y ==> zero < ?y) {
193       zero =>
194         assume is-Less := (S zero < zero)
195         let {is-not-Less := (!chain->
196           [true ==> (~ S zero < zero)
197             [Less.not-zero]])
198         (!from-complements (zero < zero) is-Less is-not-Less)
199 | (S m) =>
200   assume (S zero < S m)
201   (!chain-> [true ==> (zero < S m) [Less.zero<S]])
202 }
203 | (S n) =>
204   let {induction-hypothesis := (forall ?y . S n < ?y ==> n < ?y)}
205   datatype-cases (forall ?y . S S n < ?y ==> S n < ?y) {
206     zero =>
207       assume Less := (S S n < zero)
208       let {not-Less := (!chain->

```

```

209             [true ==> (~ S S n < zero)
210               [Less.not-zero]]}
211     (!from-complements (S n < zero) Less not-Less)
212 | (S m) =>
213   (!chain [(S S n < S m)
214            ==> (S n < m)           [Less.injective]
215            ==> (n < m)             [induction-hypothesis]
216            ==> (S n < S m)         [Less.injective]])
217   }
218 }
219
220 datatype-cases Less.S2 {
221   zero =>
222     conclude (forall ?y . zero < ?y ==> zero < (S ?y))
223     pick-any y
224       assume (zero < y)
225       (!chain-> [true ==> (zero < S y) [Less.zero<S]])
226 | (S m) =>
227   conclude (forall ?y . S m < ?y ==> S m < (S ?y))
228   pick-any y
229     (!chain [(S m < y)
230             ==> (m < y)           [Less.S1]
231             ==> (S m < S y)       [Less.injective]])
232 }
233
234 by-induction Less.S-step {
235   zero =>
236     datatype-cases
237     (forall ?y . zero < (S ?y) & zero /= ?y ==> zero < ?y) {
238     zero =>
239       assume (zero < S zero & zero /= zero)
240       (!from-complements (zero < zero) (!reflex zero) (zero /= zero))
241 | (S y) =>
242   assume (zero < (S S y) & zero /= S y)
243   (!chain-> [true ==> (zero < S y) [Less.zero<S]])
244 }
245 | (S x) =>
246   let {induction-hypothesis :=
247         (forall ?y . x < (S ?y) & x /= ?y ==> x < ?y)}
248   datatype-cases
249   (forall ?y . S x < (S ?y) & S x /= ?y ==> S x < ?y) {
250   zero =>
251     assume (S x < S zero & S x /= zero)
252     let {Less := (!chain-> [(S x < S zero)
253                            ==> (x < zero)   [Less.injective]]);
254          not-Less := (!chain->
255                      [true ==> (~ x < zero) [Less.not-zero]])}
256     (!from-complements (S x < zero) Less not-Less)
257 | (S y) =>
258   (!chain
259    [(S x < (S S y) & S x /= S y)
260     ==> (x < S y & x /= y) [Less.injective S-injective]
261     ==> (x < y)           [induction-hypothesis]
262     ==> (S x < S y)       [Less.injective]])
263 }
264 }
265
266 by-induction Less.discrete {
267   zero =>
268     (!by-contradiction (~ exists ?x . zero < ?x & ?x < S zero)
269     assume A := (exists ?x . zero < ?x & ?x < S zero)
270     pick-witness x for A witnessed
271     (!two-cases
272      assume (x = zero)
273      let {Less := (!chain->
274                  [(zero < x) ==> (zero < zero) [(x = zero)]];
275          not-Less := (!chain->
276                      [true ==> (~ zero < zero) [Less.not-zero]])}
277      (!absurd Less not-Less)
278     assume (x /= zero)

```

```

279     let {C :=
280       (!chain-> [(x != zero) ==> (exists ?m . x = S ?m)
281                 [nonzero-S]])}
282     pick-witness m for C
283     let {Less := (!chain->
284                 [(x < S zero)
285                  ==> (S m < S zero)   [(x = S m)]
286                  ==> (m < zero)       [Less.injective]])};
287     not-Less := (!chain->
288                 [true ==> (~ m < zero)
289                  [Less.not-zero]])}
290     (!absurd Less not-Less))
291 | (S n) =>
292   let {induction-hypothesis := (~ exists ?x . n < ?x & ?x < S n)}
293       (!by-contradiction (~ exists ?x . S n < ?x & ?x < S S n))
294       assume A := (exists ?x . S n < ?x & ?x < S S n)
295       pick-witness x for A witnessed
296       (!two-cases
297         assume (x = zero)
298         let {is-Less := (!chain->
299                       [(S n < x) ==> (S n < zero)
300                        [(x = zero)])];
301             is-not-Less := (!chain->
302                             [true ==> (~ S n < zero)
303                              [Less.not-zero]])}
304         (!absurd is-Less is-not-Less)
305         assume (x != zero)
306         let {C :=
307             (!chain-> [(x != zero) ==> (exists ?m . x = S ?m)
308                       [nonzero-S]])}
309             pick-witness m for C witnessed
310             let {_ := (!chain->
311                       [(S n < x)
312                        ==> (S n < S m)   [witnessed]
313                        ==> (n < m)       [Less.injective]])};
314                 E := (!chain->
315                       [(x < S S n)
316                        ==> (S m < S S n) [witnessed]
317                        ==> (m < S n)     [Less.injective]
318                        ==> (n < m & m < S n) [augment]
319                        ==> (exists ?m . n < ?m & ?m < S n)
320                          [existence]])}
321             (!absurd E induction-hypothesis))
322   }
323
324 conclude Less.transitive
325 let
326   {transitive0 :=
327     # A version with the easiest-to-induct-on variable first:
328     (forall ?x ?y ?z . ?x < ?y & ?y < ?z ==> ?x < ?z);
329   _ := by-induction transitive0 {
330     zero =>
331       pick-any x y
332       assume (x < y & y < zero)
333       let {not-Less := (!chain->
334                       [true ==> (~ y < zero) [Less.not-zero]])}
335       (!from-complements (x < zero) (y < zero) not-Less)
336   | (S n) =>
337     let {induction-hypothesis :=
338         (forall ?x ?y . ?x < ?y & ?y < n ==> ?x < n)}
339         pick-any x y
340         assume (x < y & y < S n)
341         conclude (x < S n)
342         let {_ := conclude (x < n)
343             (!two-cases
344               assume (y = n)
345               (!chain->
346                 [(x < y) ==> (x < n) [(y = n)])])
347             assume (y != n)
348             (!chain->

```

```

349                                     [(y =/= n)
350                                     ==> (y < S n & y =/= n) [augment]
351                                     ==> (y < n) [Less.S-step]
352                                     ==> (x < y & y < n) [augment]
353                                     ==> (x < n) [induction-hypothesis]]])}
354         (!chain-> [(x < n) ==> (x < S n) [Less.S2]])
355     })
356     pick-any x y z
357     (!chain [(x < y & y < z) ==> (x < z) [transitive0]])
358
359 by-induction Less.irreflexive {
360   zero => (!chain-> [true ==> (~ zero < zero) [Less.not-zero]])
361 | (S n) => (!chain-> [(~ n < n) ==> (~ S n < S n) [Less.injective]])
362 }
363
364 conclude Less.asymmetric
365   pick-any x y
366     assume (x < y)
367     (!by-contradiction (~ y < x)
368       assume (y < x)
369       let {is-Less := (!chain->
370         [(y < x)
371         ==> (x < y & y < x) [augment]
372         ==> (x < x) [Less.transitive]]);
373         is-not-Less := (!chain->
374           [true ==> (~ x < x) [Less.irreflexive]])}
375       (!absurd is-Less is-not-Less))
376
377 conclude Less.S-not-<
378   pick-any n
379     (!by-contradiction (~ S n < n)
380       assume (S n < n)
381       (!absurd
382         (!chain-> [(S n < n) ==> (n < n) [Less.S1]])
383         (!chain-> [true ==> (~ n < n) [Less.irreflexive]])))
384
385 by-induction Less.trichotomy {
386   zero =>
387     pick-any n
388       assume (~ zero < n & zero =/= n)
389       conclude (n < zero)
390       let {has-predecessor :=
391         (!chain->
392           [(zero =/= n)
393           ==> (n =/= zero) [sym]
394           ==> (exists ?k . n = (S ?k)) [nonzero-S]]);
395         pick-witness k for has-predecessor
396         let {Less := (!chain->
397           [true ==> (zero < (S k)) [Less.zero<S]
398           ==> (zero < n) [(n = (S k))]];
399         not-Less := (~ zero < n)}
400         (!from-complements (n < zero) Less not-Less)
401 | (S m) =>
402   let {induction-hypothesis :=
403     (forall ?n . (~ m < ?n & m =/= ?n) ==> ?n < m)}
404     datatype-cases (forall ?n . ~ S m < ?n & S m =/= ?n ==>
405       ?n < S m)
406     { zero =>
407       assume (~ S m < zero & S m =/= zero)
408       (!chain-> [true ==> (zero < S m) [Less.zero<S]])
409     | (S k) =>
410       assume A := (~ S m < (S k) & S m =/= (S k))
411       (!chain->
412         [A ==> (~ m < k & m =/= k) [Less.injective S-injective]
413         ==> (k < m) [induction-hypothesis]
414         ==> ((S k) < S m) [Less.injective]])
415     }
416 }
417
418 conclude Less.trichotomy1

```

```

419 pick-any m:N n
420   assume ( $\sim m < n \ \& \ \sim n < m$ )
421     (!by-contradiction (m = n)
422       (!chain
423         [(m  $\neq$  n)  $\implies$  ( $\sim m < n \ \& \ m \neq n$ ) [augment]
424            $\implies$  (n < m) [Less.trichotomy]
425            $\implies$  (n < m &  $\sim n < m$ ) [augment]
426            $\implies$  false [prop-taut]]))
427
428 conclude Less.trichotomy2
429 pick-any m:N n
430   let {A := assume (m = n)
431     let {C := (!chain->
432       [true  $\implies$  ( $\sim m < m$ ) [Less.irreflexive]])}
433     (!both (!chain-> [C  $\implies$  ( $\sim m < n$ ) [(m = n)]]
434       (!chain-> [C  $\implies$  ( $\sim n < m$ ) [(m = n)]]));
435     B := (!chain [( $\sim m < n \ \& \ \sim n < m$ )  $\implies$  (m = n)
436       [Less.trichotomy1]])}
437     (!equiv A B)
438
439 conclude Less.not-equal
440 pick-any m:N n
441   assume (m < n)
442     (!by-contradiction (m  $\neq$  n)
443       assume (m = n)
444         let {is-not-Less :=
445           (!chain->
446             [(m = n)  $\implies$  ( $\sim m < n \ \& \ \sim n < m$ ) [Less.trichotomy2]
447                $\implies$  ( $\sim m < n$ ) [left-and]])}
448           (!absurd (m < n) is-not-Less))
449
450 conclude Less.not-equal1
451 pick-any m:N n
452   assume (m < n)
453     (!by-contradiction (n  $\neq$  m)
454       assume (n = m)
455         let {is-not-Less :=
456           (!chain->
457             [m = n [n = m]
458                $\implies$  ( $\sim m < n \ \& \ \sim n < m$ ) [Less.trichotomy2]
459                $\implies$  ( $\sim (m < n)$ ) [left-and]])}
460           (!absurd (m < n) is-not-Less))
461
462 conclude Less=.Implied-by-<
463 pick-any m n
464   (!chain [(m < n)  $\implies$  (m < n | m = n) [alternate]
465      $\implies$  (m  $\leq$  n) [Less=.definition]])
466
467 conclude Less=.Implied-by-equal
468 pick-any m:N n
469   (!chain [(m = n)  $\implies$  (m < n | m = n) [alternate]
470      $\implies$  (m  $\leq$  n) [Less=.definition]])
471
472 conclude Less=.reflexive
473 pick-any n
474   let {disj := (!right-either (n < n) (!reflex n))}
475     (!chain-> [disj  $\implies$  (n  $\leq$  n) [Less=.definition]])
476
477 datatype-cases Less=.zero<= {
478   zero =>
479     (!chain->
480       [(zero = zero)  $\implies$  (zero  $\leq$  zero) [Less=.Implied-by-equal]])
481 | (S n) =>
482     (!chain->
483       [true  $\implies$  (zero < S n) [Less.zero<S]
484          $\implies$  (zero  $\leq$  S n) [Less=.Implied-by-<]])
485 }
486
487 datatype-cases Less=.S-zero-S-n {
488   zero =>

```

```

489   let {disj := (!right-either (S zero < S zero) (!reflex (S zero)))}
490     (!chain->
491       [disj ==> (S zero <= S zero) [Less=.definition]])
492 | (S m) =>
493   let {Less :=
494     (!chain->
495       [true ==> (zero < S m) [Less.zero<S]
496         ==> (S zero < (S S m)) [Less.injective]]);
497     disj := (!left-either Less (S zero = (S S m)))}
498     (!chain->
499       [disj ==> (S zero <= (S S m)) [Less=.definition]])
500 }
501
502 conclude Less=.injective
503 pick-any n m
504   (!chain
505     [(S n <= S m)
506       <==> (S n < S m | S n = S m) [Less=.definition]
507       <==> (n < m | n = m) [Less.injective S-injective]
508       <==> (n <= m) [Less=.definition]])
509
510 conclude Less=.not-S
511 pick-any n
512   (!by-contradiction (~ S n <= n)
513     assume hyp := (S n <= n)
514     let {disjunction :=
515       (!chain->
516         [hyp ==> (S n < n | S n = n) [Less=.definition]])}
517     (!cases disjunction
518       assume hyp1 := (S n < n)
519       let {not-hyp1 := (!chain-> [true ==> (~ hyp1)
520         [Less.S-not-<]])}
521         (!absurd hyp1 not-hyp1)
522       assume hyp2 := (S n = n)
523       let {not-hyp2 := (!chain->
524         [true ==> (~ hyp2) [S-not-same]])}
525         (!absurd hyp2 not-hyp2)))
526
527 conclude Less=.S-not-equal
528 pick-any k:N n
529   assume hyp := (S k <= n)
530   let {P := (S n <= n)}
531     (!by-contradiction (k /= n)
532       assume (k = n)
533         (!absurd
534           (!chain-> [hyp ==> P [(k = n)]])
535           (!chain-> [true ==> (~ P) [Less=.not-S]])))
536
537 conclude Less=.discrete
538 pick-any m n
539   assume (m < n)
540   (!by-contradiction (S m <= n)
541     assume p := (~ S m <= n)
542     let {in-between := (exists ?n . m < ?n & ?n < S m)}
543       (!absurd
544         (!chain->
545           [p ==> (~ (S m < n | S m = n)) [Less=.definition]
546             ==> (~ S m < n & S m /= n) [dm]
547             ==> (n < S m) [Less.trichotomy]
548             ==> (m < n & n < S m) [augment]
549             ==> in-between [existence]])
550         (!chain->
551           [true ==> (~ in-between) [Less.discrete]])))
552
553 conclude Less=.trichotomy1
554 pick-any m n
555   (!chain [(~ n <= m)
556     ==> (~ (n < m | n = m)) [Less=.definition]
557     ==> (~ n < m & ~ (n = m)) [dm]
558     ==> (m < n) [Less.trichotomy]])

```



```

559
560 conclude Less=.trichotomy2
561   pick-any m n
562     (!contra-pos (!instance Less=.trichotomy1 [n m]))
563
564 conclude Less=.trichotomy3
565   pick-any m:N n
566     assume h1 := (n < m)
567     (!by-contradiction (~ m <= n)
568       assume h2 := (m <= n)
569       let {h3 := (!chain->
570         [h2 ==> (m < n | m = n) [Less=.definition]]);
571         selfLess :=
572           conclude (n < n)
573             (!cases h3
574               assume h4 := (m < n)
575                 (!chain-> [h1 ==> (h1 & h4)      [augment]
576                   ==> (n < n)      [Less.transitive]])
577                 assume (m = n)
578                   (!chain-> [(n < m) ==> (n < n) [(m = n)]]));
579         not-selfLess :=
580           (!chain->
581             [true ==> (~ n < n) [Less.irreflexive]])}
582     (!absurd selfLess not-selfLess)
583
584 conclude Less=.transitive
585   pick-any x:N y:N z
586     assume (x <= y & y <= z)
587     let {disj1 := (!chain->
588       [(y <= z) ==> (y < z | y = z) [Less=.definition]]}
589     conclude (x <= z)
590     (!cases disj1
591       assume C := (y < z)
592       let {disj2 := (!chain->
593         [(x <= y)
594           ==> (x < y | x = y) [Less=.definition]]}
595       conclude (x <= z)
596       (!cases disj2
597         assume (x < y)
598           (!chain->
599             [(x < y) ==> (x < y & C) [augment]
600               ==> (x < z) [Less.transitive]
601                 ==> (x <= z) [Less.Implied-by-<]])
602         assume (x = y)
603           (!chain-> [(y < z)
604             ==> (x < z) [(x = y)]
605               ==> (x <= z) [Less.Implied-by-<]])
606         assume (y = z)
607           (!chain-> [(x <= y) ==> (x <= z) [(y = z)]]))
608
609 conclude Less=.transitive2
610   pick-any x:N y:N z:N
611     assume (x <= y & y < z)
612     conclude (x < z)
613     let {D := (!chain->
614       [(x <= y) ==> (x < y | x = y) [Less=.definition]]}
615     (!cases D
616       assume (x < y)
617         (!chain-> [(x < y & y < z) ==> (x < z) [Less.transitive]])
618       assume (x = y)
619         (!chain-> [(y < z) ==> (x < z) [(x = y)]]))
620
621 conclude Less=.transitive1
622   pick-any x:N y:N z:N
623     assume (x < y & y <= z)
624     (!by-contradiction (x < z)
625       assume A := (~ x < z)
626       (!absurd
627         (y <= z)
628         conclude (~ y <= z)

```

```

629         (!chain->
630           [A ==> (z <= x)      [Less=.trichotomy2]
631             ==> (z <= x & x < y) [augment]
632             ==> (z < y)      [Less=.transitive2]
633             ==> (~ y <= z)   [Less=.trichotomy3]]))
634
635 conclude Less=.S1
636 pick-any n:N m
637   assume hyp := (n <= m)
638   let {disj := (!chain->
639     [hyp ==> (n < m | n = m) [Less=.definition]])}
640   conclude (n < S m)
641     (!cases disj
642       assume p := (n < m)
643         (!chain->
644           [true ==> (m < S m)      [Less.<S]
645             ==> (p & m < S m)    [augment]
646             ==> (n < S m)      [Less.transitive]])
647         assume (n = m)
648           (!chain-> [true ==> (m < S m) [Less.<S]
649             ==> (n < S m) [(n = m)]]))
650
651 conclude Less=.S2
652 pick-any n m
653   (!chain [(n <= m) ==> (n < S m) [Less=.S1]
654     ==> (n <= S m) [Less=.Implied-by-<]])
655
656 conclude Less=.S3
657 pick-any n
658   (!chain->
659     [true ==> (n < S n) [Less.<S]
660     ==> (n <= S n) [Less=.Implied-by-<]])
661
662 conclude Less=.trichotomy4
663 pick-any m n
664   (!contra-pos (!instance Less=.trichotomy3 [n m]))
665
666 conclude Less=.trichotomy5
667 pick-any m:N n
668   assume (m <= n & n <= m)
669   (!by-contradiction (m = n)
670     assume (m /= n)
671     let {P1 := (!chain->
672       [(m <= n)
673         ==> (m < n | m = n) [Less=.definition]]);
674       P2 := (!chain->
675         [(n <= m)
676           ==> (n < m | n = m) [Less=.definition]])}
677     (!cases P1
678       assume (m < n)
679         (!cases P2
680           (!chain
681             [(n < m)
682               ==> (m < n & n < m) [augment]
683               ==> (~ n < m & n < m) [Less.asymmetric]
684               ==> false [prop-taut]])
685           assume (n = m)
686             (!absurd (!sym (n = m)) (m /= n)))
687           assume (m = n)
688             (!absurd (m = n) (m /= n))))
689
690 by-induction Less.Plus-cancellation {
691   zero =>
692     pick-any m n
693     (!chain [(m + zero < n + zero)
694       ==> (m < n) [Plus.right-zero]])
695 | (S k) =>
696   let {induction-hypothesis :=
697     (forall ?m ?n . ?m + k < ?n + k ==> ?m < ?n)}
698   pick-any m n

```

```

699     (!chain
700       [(m + S k < n + S k)
701        ==> (S (m + k) < S (n + k)) [Plus.right-nonzero]
702         ==> (m + k < n + k)       [Less.injective]
703         ==> (m < n)                [induction-hypothesis]])
704   }
705
706 conclude Less=.Plus-cancellation
707 pick-any k m n
708 assume p1 := (m + k <= n + k)
709 let {p2 :=
710   (!chain-> [p1 ==> (m + k < n + k | m + k = n + k)
711             [Less=.definition]])}
712 conclude (m <= n)
713   (!cases p2
714     (!chain [(m + k < n + k)
715              ==> (m < n)           [Less.Plus-cancellation]
716              ==> (m <= n)         [Less=.Implied-by-<]])
717     (!chain [(m + k = n + k)
718              ==> (m = n)           [Plus.=cancellation]
719              ==> (m <= n)         [Less=.Implied-by-equal]]))
720
721 conclude Less.Plus-k
722 pick-any k m n
723 assume hyp1 := (m < n)
724 let {goal := (m + k < n + k)}
725   (!by-contradiction goal
726     (!chain
727       [(~ goal) ==> (n + k <= m + k) [Less.trichotomy2]
728        ==> (n <= m)                 [Less.Plus-cancellation]
729        ==> (~ m < n)                 [Less.trichotomy4]
730        ==> (hyp1 & ~ m < n) [augment]
731        ==> false                    [prop-taut]]))
732
733 # Alternative version, without using taut [K]:
734
735 conclude Less.Plus-k
736 pick-any k m n
737 assume hyp1 := (m < n)
738 let {goal := (m + k < n + k)}
739   (!by-contradiction goal
740     (!chain
741       [(~ goal) ==> (n + k <= m + k) [Less.trichotomy2]
742        ==> (n <= m)                 [Less.Plus-cancellation]
743        ==> (~ m < n)                 [Less.trichotomy4]
744        ==> false                    [method (p) (!absurd hyp1 p)]]))
745
746 by-induction Less.Plus-k1 {
747   zero =>
748     pick-any m n
749     (!chain [(m < n) ==> (m < n + zero) [Plus.right-zero]])
750 | (S k) =>
751   let {IH := (forall ?m ?n . ?m < ?n ==> ?m < ?n + k)}
752     pick-any m n
753     assume (m < n)
754     conclude (m < n + (S k))
755     (!chain->
756       [true ==> (k < (S k))           [Less.<S]
757        ==> (k + n < (S k) + n)       [Less.Plus-k]
758        ==> (n + k < n + (S k))       [Plus.commutative]
759        ==> (m < n & n + k < n + (S k)) [augment]
760        ==> (m < n + k & n + k < n + (S k)) [IH]
761        ==> (m < n + (S k))           [Less.transitive]])
762 }
763
764 conclude Less=.Plus-k
765 pick-any k m n
766 assume hyp1 := (m <= n)
767 let {goal := (m + k <= n + k)}
768   (!by-contradiction goal

```

```

769     (!chain [(~ goal) ==> (n + k < m + k) [Less=.trichotomy1]
770             ==> (n < m) [Less.Plus-cancellation]
771             ==> (~ m <= n) [Less=.trichotomy3]
772             ==> (hyp1 & ~ m <= n) [augment]
773             ==> false [prop-taut]]))
774
775 by-induction Less=.Plus-k1 {
776   zero =>
777     pick-any m n
778     (!chain [(m <= n) ==> (m <= n + zero) [Plus.right-zero]])
779 | (S k) =>
780   let {IH := (forall ?m ?n . ?m <= ?n ==> ?m <= ?n + k)}
781   pick-any m n
782   assume (m <= n)
783   conclude (m <= n + (S k))
784   (!chain->
785     [true ==> (k <= (S k)) [Less=.S3]
786              ==> (k + n <= (S k) + n) [Less=.Plus-k]
787              ==> (n + k <= n + (S k)) [Plus.commutative]
788              ==> (m <= n & n + k <= n + (S k)) [augment]
789              ==> (m <= n + k & n + k <= n + (S k)) [IH]
790              ==> (m <= n + (S k)) [Less=.transitive]])
791 }
792
793 conclude Less.Plus-k-equiv
794 pick-any k m n
795 (!equiv (!chain [(m < n) ==> (m + k < n + k) [Less.Plus-k]]
796               (!chain [(m + k < n + k) ==> (m < n) [Less.Plus-cancellation]])))
797
798 by-induction Less=.k-Less= {
799   zero =>
800     conclude (forall ?m ?n . ?n = ?m + zero ==> ?m <= ?n)
801     pick-any m n
802     assume hyp := (n = m + zero)
803     (!chain->
804       [m = (m + zero) [Plus.right-zero]
805        = n [hyp]
806        ==> (m <= n) [Less=.Implied-by-equal]])
807 | (S k) =>
808   conclude (forall ?m ?n . ?n = ?m + (S k) ==> ?m <= ?n)
809   pick-any m n
810   let {ind-hyp := (forall ?m ?n . ?n = ?m + k ==> ?m <= ?n)}
811   assume hyp := (n = m + (S k))
812   let {C := (!chain->
813             [n = (m + (S k)) [hyp]
814              = (S (m + k)) [Plus.right-nonzero]
815              = (S m + k) [Plus.left-nonzero]
816              ==> (S m <= n) [ind-hyp]])}
817   (!chain->
818     [true ==> (m <= S m) [Less=.S3]
819              ==> ((m <= S m) & C) [augment]
820              ==> (m <= n) [Less=.transitive]])
821 }
822
823 conclude Less=.zero2
824 pick-any n
825 assume hyp := (n <= zero)
826 let {disj := (!chain->
827               [hyp ==> (n < zero | n = zero) [Less=.definition]])}
828 (!cases disj
829   assume (n < zero)
830   let {not-Less := (!chain->
831                   [true ==> (~ n < zero) [Less.not-zero]])}
832   (!from-complements (n = zero) (n < zero) not-Less)
833   assume (n = zero)
834   (!claim (n = zero)))
835
836 # Alternative - and much shorter - version using chaining and disjunctive
837 # syllogism. (This essentially lifts the case analysis from the
838 # source text and relegates it to the dsyl method.) [K]:

```

```

839
840 conclude Less=.zero2
841   pick-any n
842     assume hyp := (n <= zero)
843     (!dsyl (!chain->
844       [hyp ==> (n < zero | n = zero) [Less=.definition]])
845       (!chain->
846         [true ==> (~ n < zero)           [Less.not-zero]]))
847
848 conclude Less=.not-S-zero
849   pick-any n
850     (!by-contradiction (~ S n <= zero)
851       assume A := (S n <= zero)
852       (!absurd
853         (!chain->
854           [A ==> (S n = zero) [Less=.zero2]])
855         (!chain->
856           [true ==> (S n /= zero) [S-not-zero]])))
857
858 conclude Less.Reverse-S
859   pick-any n m
860     (!chain [(~ m < n)
861       ==> (n <= m)           [Less=.trichotomy2]
862       ==> (n < S m)         [Less=.S1]])
863
864 conclude Less.S4
865 conclude (forall ?m ?n . S ?m < ?n ==> (exists ?n' . ?n = (S ?n')))
866 pick-any m n
867   assume A := (S m < n)
868   let {B := (!chain->
869     [true ==> (n = zero | (exists ?n' . n = (S ?n')))
870     [(!constructor-exhaustiveness "N")]]})
871     (!cases B
872       assume C := (n = zero)
873       (!from-complements (exists ?n' . n = (S ?n'))
874         (!chain-> [A ==> (S m < zero) [C]])
875         (!chain-> [true ==> (~ S m < zero)
876           [Less.not-zero]]))
877       assume D := (exists ?n' . n = (S ?n'))
878       (!claim D))
879
880 conclude Less=.S4
881 conclude (forall ?m ?n . S ?m <= ?n ==> (exists ?n' . ?n = S ?n'))
882 pick-any m n
883   assume A := (S m <= n)
884   let {B := (!chain->
885     [A ==> (S m < n | S m = n) [Less=.definition]])}
886     (!cases B
887       (!chain
888         [(S m < n) ==> (exists ?n' . n = (S ?n')) [Less.S4]])
889       (!chain
890         [(S m = n) ==> (n = S m) [sym]
891           ==> (exists ?n' . n = (S ?n')) [existence]]))
892
893 conclude Less=.S5
894 conclude (forall ?n ?m . ?n <= S ?m & ?n /= S ?m ==> ?n <= ?m)
895 pick-any n m
896   let {A1 := (n <= S m);
897     A2 := (n /= S m)}
898   assume (A1 & A2)
899   let {B := (!chain->
900     [A1 ==> (n < S m | n = S m) [Less=.definition]])}
901     (!cases B
902       assume B1 := (n < S m)
903       (!chain->
904         [B1 ==> (S n <= S m) [Less=.discrete]
905           ==> (n <= m) [Less=.injective]])
906       assume B2 := (n = S m)
907       (!from-complements (n <= m) B2 A2))
908

```

```

909 conclude Less.=zero
910   pick-any m
911     (!chain
912       [(m < one)
913        ==> (~ one <= m)           [Less=.trichotomy3]
914         ==> (~ S zero <= m)      [one-definition]
915         ==> (~ zero < m)        [Less=.discrete]
916         ==> (m = zero)          [Less.=zero]])
917
918 conclude Less=.zero<=one
919   pick-any m
920     assume (m = zero)
921       conclude (m <= one)
922         (!chain->
923           [true ==> (zero <= one) [Less=.zero<=]
924            ==> (m <= one)      [m = zero]])
925
926 extend-module Times {
927
928 define ==cancellation :=
929   (forall y z x . zero < x & x * y = x * z ==> y = z)
930
931 by-induction ==cancellation {
932   zero =>
933     pick-any z x
934       assume (zero < x & x * zero = x * z)
935         conclude (zero = z)
936           let {D := (!chain-> [(x * z)
937                               = (x * zero)   [(x * zero = x * z)]
938                               = zero         [right-zero]
939                               ==> (x = zero | z = zero)
940                                   [no-zero-divisors]])}
941             (!cases D
942               assume (x = zero)
943                 (!from-complements (zero = z)
944                   (x = zero)
945                     (!chain->
946                       [(zero < x) ==> (x /= zero) [Less.not-equal]])
947                     assume (z = zero)
948                       (!sym (z = zero)))
949 | (S y) =>
950   let {ind-hyp := (forall ?z ?x . zero < ?x & ?x * y = ?x * ?z ==> y = ?z)}
951     datatype-cases (forall ?z ?x .
952       zero < ?x & ?x * (S y) = ?x * ?z ==> (S y) = ?z)
953     {
954       zero =>
955         conclude (forall ?x .
956           zero < ?x & ?x * (S y) = ?x * zero ==> (S y) = zero)
957         pick-any x
958           assume (zero < x & x * (S y) = x * zero)
959             let {C1 := (!chain->
960               [(x * (S y))
961                = (x * zero) [(x * (S y) = x * zero)]
962                = zero       [right-zero]
963                ==> (x = zero | (S y) = zero)
964                    [no-zero-divisors]])}
965               (!cases C1
966                 assume (x = zero)
967                   (!from-complements ((S y) = zero)
968                     (x = zero)
969                       (!chain->
970                         [(zero < x) ==> (x /= zero) [Less.not-equal]])
971                       assume ((S y) = zero)
972                         (!claim ((S y) = zero)))
973 | (S z) =>
974   conclude (forall ?x . zero < ?x & ?x * (S y) = ?x * (S z)
975     ==> (S y) = (S z))
976   pick-any x
977     assume (zero < x & x * (S y) = x * (S z))
978     (!chain->

```

```

979         [(x * y + x)
980          = (x * (S y)) [right-nonzero]
981          = (x * (S z)) [(x * (S y) = x * (S z))]
982          = (x * z + x)
983          ==> (x * y = x * z) [Plus.-=cancellation]
984          ==> (zero < x & x * y = x * z) [augment]
985          ==> (y = z) [ind-hyp]
986          ==> (S y = S z) [S-injective]])
987     }
988 }
989
990 define <-cancellation :=
991   (forall y z x . zero < x & x * y < x * z ==> y < z)
992
993 by-induction <-cancellation {
994   zero =>
995     pick-any z x
996     assume (zero < x & x * zero < x * z)
997     (!by-contradiction (zero < z)
998       assume A := (~ zero < z)
999       let {_ := (!chain-> [A ==> (z = zero) [Less.=zero]])}
1000       (!absurd
1001         (!chain->
1002           [(x * zero < x * z)
1003            ==> (zero < zero) [right-zero (z = zero)]]
1004           (!chain->
1005             [true ==> (~ zero < zero) [Less.irreflexive]])))
1006 | (S y) =>
1007   let {ind-hyp := (forall ?z ?x . zero < ?x & ?x * y < ?x * ?z ==> y < ?z)}
1008   datatype-cases (forall ?z ?x . zero < ?x & ?x * (S y) < ?x * ?z
1009     ==> (S y) < ?z)
1010   {
1011     zero =>
1012       pick-any x
1013       assume (zero < x & x * (S y) < x * zero)
1014       (!from-complements ((S y) < zero)
1015         (!chain-> [(x * (S y) < x * zero)
1016                   ==> (x * (S y) < zero) [right-zero]]
1017         (!chain-> [true ==> (~ x * (S y) < zero) [Less.not-zero]]))
1018 | (S z) =>
1019     pick-any x
1020     assume (zero < x & x * (S y) < x * (S z))
1021     conclude ((S y) < (S z))
1022     (!chain->
1023       [(x * (S y) < x * (S z))
1024        ==> (x * y + x < x * z + x) [right-nonzero]
1025        ==> (x * y < x * z) [Less.Plus-cancellation]
1026        ==> (y < z) [(zero < x) ind-hyp]
1027        ==> ((S y) < (S z)) [Less.injective]])
1028   }
1029 }
1030
1031 define <-cancellation-conv :=
1032   (forall x y z . zero < x & y < z ==> x * y < x * z)
1033
1034 conclude <-cancellation-conv
1035 pick-any x y z
1036 assume A1 := (zero < x & y < z)
1037 let {goal := (x * y < x * z)}
1038 (!by-contradiction goal
1039   assume (~ goal)
1040   let {D := (!chain->
1041     [ (~ goal)
1042      ==> (x * z <= x * y) [Less=.trichotomy2]
1043      ==> (x * z < x * y | x * z = x * y) [Less=.definition] ]})
1044     (!cases D
1045       assume A2 := (x * z < x * y)
1046       (!chain->
1047         [A2 ==> (z < y) [<-cancellation]

```

```

1049             ==> (~ y < z)           [Less.asymmetric]
1050             ==> (y < z & ~ y < z)    [augment]
1051             ==> false                [prop-taut]]
1052         assume A3 := (x * z = x * y)
1053         (!absurd
1054         (!chain->
1055         [(zero < x & A3) ==> (z = y) [=-cancellation]])
1056         (!chain-> [(y < z)
1057         ==> (~ z = y) [Less.not-equal]])))
1058
1059 define <=-cancellation-conv :=
1060   (forall x y z . y <= z ==> x * y <= x * z)
1061
1062
1063 conclude <=-cancellation-conv
1064 pick-any x y z
1065   assume A := (y <= z)
1066   let {goal := (x * y <= x * z)}
1067     (!two-cases
1068     assume A1 := (zero < x)
1069     (!by-contradiction goal
1070     assume (~ goal)
1071     (!chain->
1072     [(~ goal)
1073     ==> (x * z < x * y) [Less.trichotomy1]
1074     ==> (z < y) [A1 <-cancellation]
1075     ==> (~ y <= z) [Less.trichotomy4]
1076     ==> (A & ~ A) [augment]
1077     ==> false [prop-taut]]))
1078     assume A2 := (~ zero < x)
1079     let {C := (!chain->
1080     [true ==> (~ x < zero) [Less.not-zero]
1081     ==> (~ x < zero & A2) [augment]
1082     ==> (x = zero) [Less.trichotomy1]]})
1083     (!chain<- [goal <== (zero * y <= zero * z) [C]
1084     <== (zero <= zero) [left-zero]
1085     <== true [Less.reflexive]])
1086
1087 define identity-lemmal :=
1088   (forall x y . zero < x & x * y = x ==> y = one)
1089 define identity-lemma2 :=
1090   (forall x y . x * y = one ==> x = one)
1091
1092 conclude identity-lemmal
1093 pick-any x y
1094   assume (zero < x & x * y = x)
1095   (!chain->
1096   [(x * y = x)
1097   ==> (x * y = x * one) [right-one]
1098   ==> (y = one) [(zero < x) ==-cancellation]])
1099
1100 conclude identity-lemma2
1101 pick-any x y
1102   assume A := (x * y = one)
1103   let {C1 := (!by-contradiction (x /= zero)
1104   assume (x = zero)
1105   (!absurd
1106   (!chain->
1107   [true ==> (zero * y = zero) [left-zero]
1108   ==> (x * y = zero) [(x = zero)]
1109   ==> (one = zero) [A]])
1110   (!chain->
1111   [true ==> (one /= zero) [one-not-zero]]))});
1112   C2 := (!by-contradiction (y /= zero)
1113   assume (y = zero)
1114   (!absurd
1115   (!chain->
1116   [true ==> (x * zero = zero) [right-zero]
1117   ==> (x * y = zero) [(y = zero)]
1118   ==> (one = zero) [A]])

```



```

1119         (!chain->
1120         [true ==> (one /= zero)      [one-not-zero]]));
1121 C3 := (!by-contradiction (~ one < x)
1122       assume (one < x)
1123       let { _ := (!chain->
1124                 [C2 ==> (zero < y)  [Less.zero<]]})
1125         (!absurd
1126         (!chain->
1127         [(one < x)
1128          ==> (zero < y & one < x)      [augment]
1129          ==> (y * one < y * x)         [<-cancellation-conv]
1130          ==> (one * y < x * y)         [commutative]
1131          ==> (one * y < one)           [A]
1132          ==> (y < (S zero))            [left-one one-definition]
1133          ==> (y < (S zero) & y /= zero) [augment]
1134          ==> (y < zero)                [Less.S-step]])
1135         (!chain->
1136         [true ==> (~ y < zero) [Less.not-zero]]));
1137 C4 := (!chain->
1138       [C3 ==> (~ (S zero) < x) [one-definition]
1139        ==> (x <= (S zero))     [Less=.trichotomy2]
1140        ==> (x < (S zero) | x = (S zero)) [Less=.definition]]})
1141 (!by-contradiction (x = one)
1142   assume A := (x /= one)
1143   (!absurd
1144   (!chain->
1145   [A ==> (C4 & A)                [augment]
1146    ==> (C4 & x /= (S zero))      [one-definition]
1147    ==> (x < (S zero))            [prop-taut]
1148    ==> (x /= zero & x < (S zero)) [augment]
1149    ==> (x < zero)                [Less.S-step]))
1150   (!chain->
1151   [true ==> (~ x < zero)         [Less.not-zero]]))
1152
1153 define squeeze :=
1154   (forall x y . x * y < x ==> y = zero)
1155
1156 conclude squeeze
1157 pick-any x:N y:N
1158   assume (x * y < x)
1159   (!by-contradiction (y = zero)
1160    assume (y /= zero)
1161     let {subgoal := (~ x * y < x);
1162         B := datatype-cases subgoal on y {
1163           zero => (!from-complements (~ x * zero < x)
1164                 (!reflex zero)
1165                 (!chain->
1166                 [(y /= zero)
1167                  ==> (zero /= zero) [(y = zero)]])])
1168           | (S y) =>
1169             (!chain->
1170             [true
1171              ==> (~ x * y < zero)      [Less.not-zero]
1172              ==> (~ x * y + x < zero + x) [Less.Plus-k-equiv]
1173              ==> (~ x * y + x < x)      [Plus.left-zero]
1174              ==> (~ x * (S y) < x)      [Times.right-nonzero]]
1175             )}
1176     (!absurd (x * y < x) subgoal))
1177 } # Times
1178
1179 } # N
1180
1181 extend-module N {
1182
1183 define combine-inequalities :=
1184   method (L)
1185     match L {
1186       (list-of (Less= x y) rest) =>
1187         match (!combine-inequalities rest) {
1188           (Less= y z) =>

```

```
1189         (!chain->
1190          [(x <= y & y <= z) ==> (x <= z) [Less=.transitive]])
1191     | true => (!claim (Less= x y))
1192   }
1193 | [] => (!true-intro)
1194 }
1195
1196 } # N
```